Refactoring with Synthesis

Veselin Raychev · Max Schäfer ·Manu Sridharan · Martin Vechev

presenter name(s) removed for FERPA considerations

Introduction

Code Refactoring - Norm vs Available tools

Code refactoring is the process of improving the structure of existing code without changing its external behavior

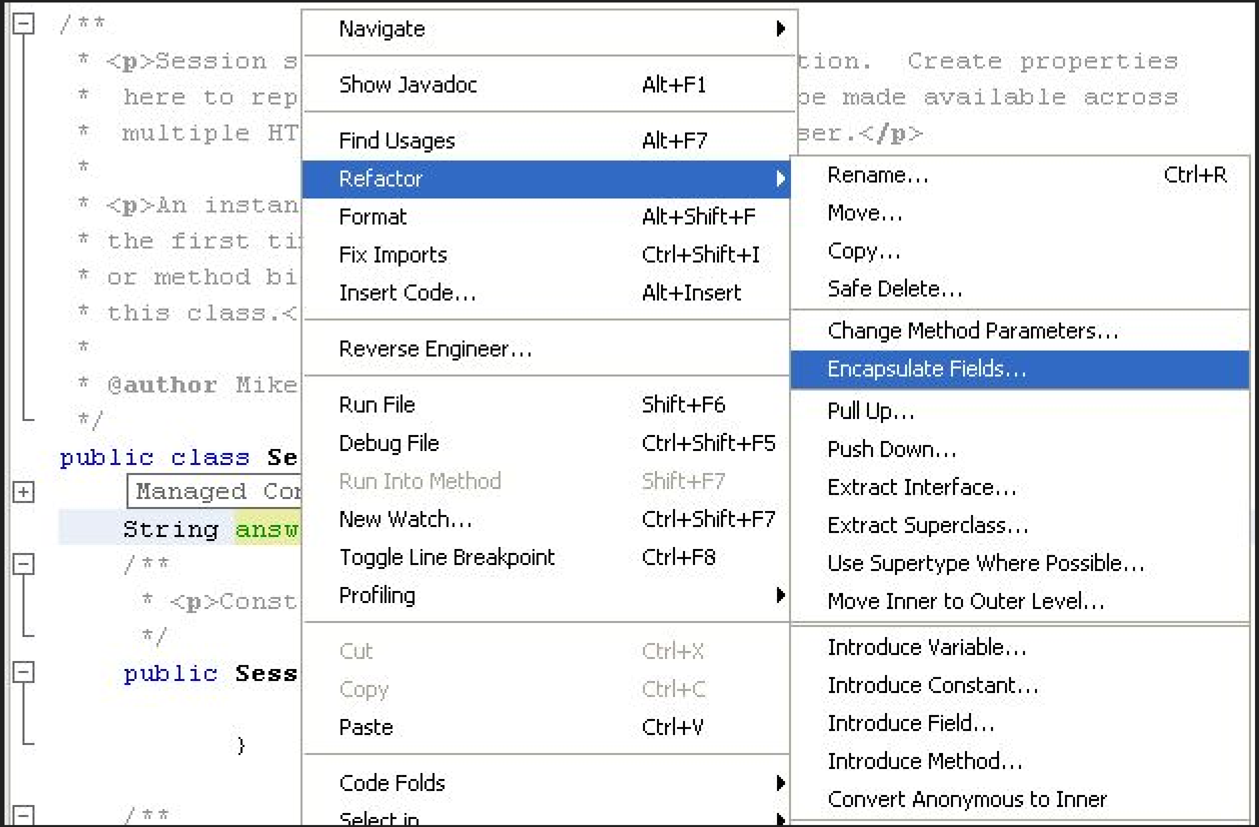
For the most part refactoring is done by hand by developers

However modern IDEs like Eclipse and IntelliJ IDEA for OO languages all provide a large number of built-in refactorings that can be activated through a menu item.

Why do developers still refactor by hand?

Developers still prefer to refactor by hand because:

- The refactorings provided by IDEs require developers to memorize names and meanings
- They are hard-coded, restrictive and inflexible
- Worst of all, they are slow and time consuming


Navigate
Show Javadoc Alt+F1
Find Usages Alt+F7
Refactor
Format Alt+Shift+F
Fix Imports Ctrl+Shift+I
Insert Code... Alt+Insert
Reverse Engineer...
Run File Shift+F6
Debug File Ctrl+Shift+F5
Run Into Method Shift+F7
New Watch... Ctrl+Shift+F7
Toggle Line Breakpoint Ctrl+F8
Profiling
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Code Folds
Rename... Ctrl+R
Move...
Copy...
Safe Delete...
Change Method Parameters...
Encapsulate Fields...
Pull Up...
Push Down...
Extract Interface...
Extract Superclass...
Use Supertype Where Possible...
Move Inner to Outer Level...
Introduce Variable...
Introduce Constant...
Introduce Field...
Introduce Method...
Convert Anonymous to Inner

RESYNTH

Veselin Raychev · Max Schäfer ·Manu Sridharan · Martin Vechev

Report by Timothy Addai, Thanh Pham, Kevin Silva

Contribution

❖ RESYNTH

A new approach to refactoring that simplifies automated refactoring to a 3-step process

★ The programmer indicates the start of a code refactoring phase.
★ Then she performs some of the code changes manually
★ Then she asks RESYNTH to complete the refactoring

RESYNTH Approach - A Quick Overview

- RESYNTH first extracts the difference between the modified program and the original program

- It then synthesizes the sequence of refactorings that achieves the desired code changes

- In order to be scalable,RESYNTH discovers a refactoring sequence within a small section of code and then extrapolates it into a full refactoring sequences

An Example

```java
public class Account {
  private String name;

  void printOwing() {
    printBanner();
    System.out.println("name:  " + name);
    System.out.println("outstanding:  " +
                       getOutstanding());
  }

  private double getOutstanding() {
    //...
  }

  private void printBanner() {
    //...
  }
}
```

(a)

```java
public class Account {
  private String name;

  void printOwing() {
    printBanner();
    printDetails(getOutstanding());
  }

  private void printDetails(double outstanding) {
    System.out.println("name:  " + name);
    System.out.println("amount:  " + outstanding);
  }

  private double getOutstanding() {
    //...
  }

  private void printBanner() {
    //...
  }
}
```

(b)

Figure 1. Example of a complicated refactoring that cannot be achieved in a single step in current IDEs; changes highlighted.

The Default Eclipse IDE Approach

- Eclipse will first use EXTRACT method on line 6 and 7 below to create printDetails()

```
6       System.out.println("name:  " + name);
7       System.out.println("outstanding:  " +
8                           getOutstanding());
```

```
private void printDetails() {
  System.out.println("name:  " + name);
  System.out.println("outstanding:  " +
                      getOutstanding());
}
```

- Then the developer would have to use the INTRODUCE PARAMETER call to create the final result shown here

```
private void printDetails(double outstanding) {
  System.out.println("name:  " + name);
  System.out.println("amount:  " + outstanding);
}
```

What’s wrong with the Eclipse Approach?

- The INTRODUCE PARAMETER call would have to be invoked by the developer but this feature is not well known by developers

- A case study by Abadi et al. showed that the EXTRACT METHOD of Eclipse worked automatically only 3 out of 13 times.

RESYNTH - A better approach

Three simple steps

- Programmer hits start on RESYNTH interface
- The programmer would then simply replace lines 6 and 7 with *printDetails(getOutstanding)*

```
6        System.out.println("name: " + name);
7        System.out.println("outstanding: " +
8                              getOutstanding();
```

```
6 printDetails(getOutstanding)
7
8
```

RESYNTH - A better approach

- Then the programmer will hit the complete refactor button which will let RESYNTH come up with the *printDetails(getOutstanding)* method shown below.

```
private void printDetails(double outstanding) {
    System.out.println("name: " + name);
    System.out.println("amount: " + outstanding);
}
```

Research Questions

- Can RESYNTH successfully perform individual refactorings?
- Will RESYNTH be able to synthesize complex refactoring sequences required for real-world complex code base?
- How would programmers feel about synthesis-based refactoring using RESYNTH?

Key Ideas

The sequences of refactoring

The process of Resynth

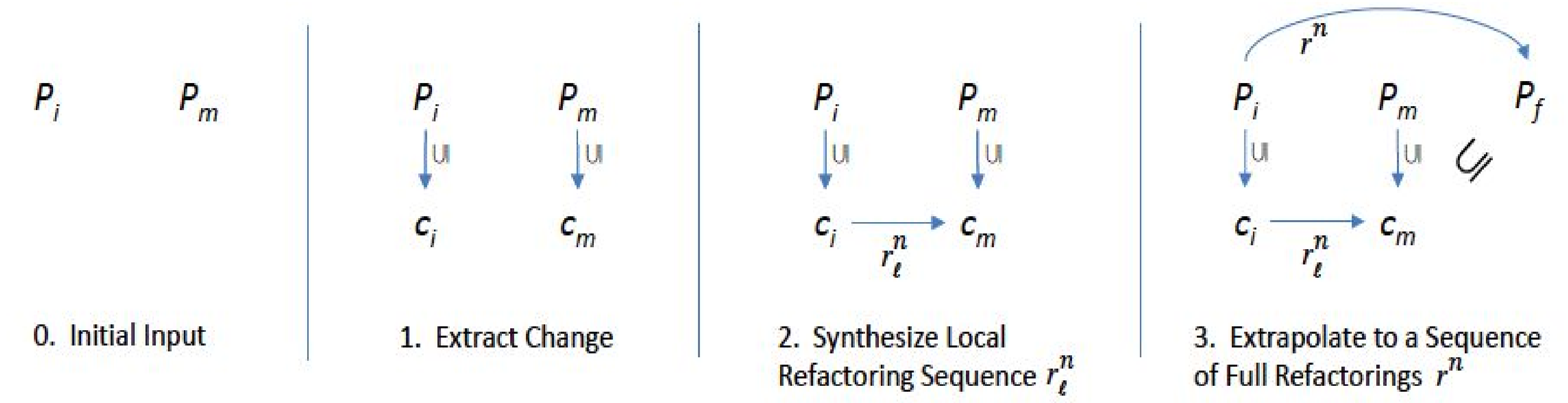


Figure 2. Synthesis Steps

Initial Input

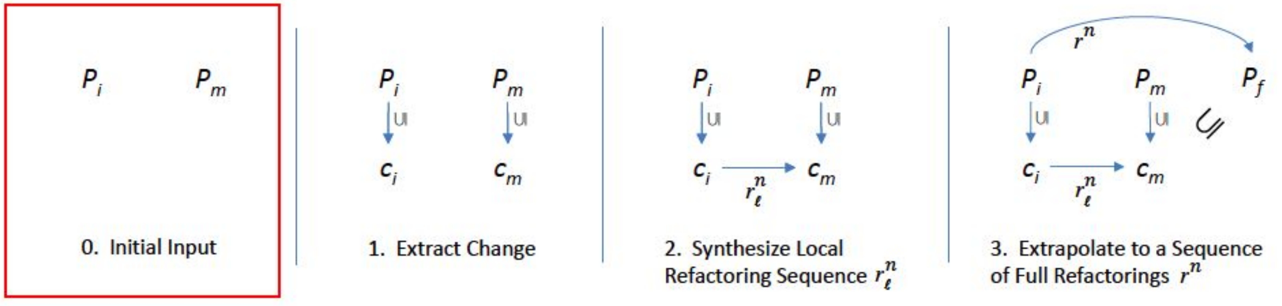


Figure 2. Synthesis Steps

- Initial Program Pi
- Modified Program Pm

Extract Change

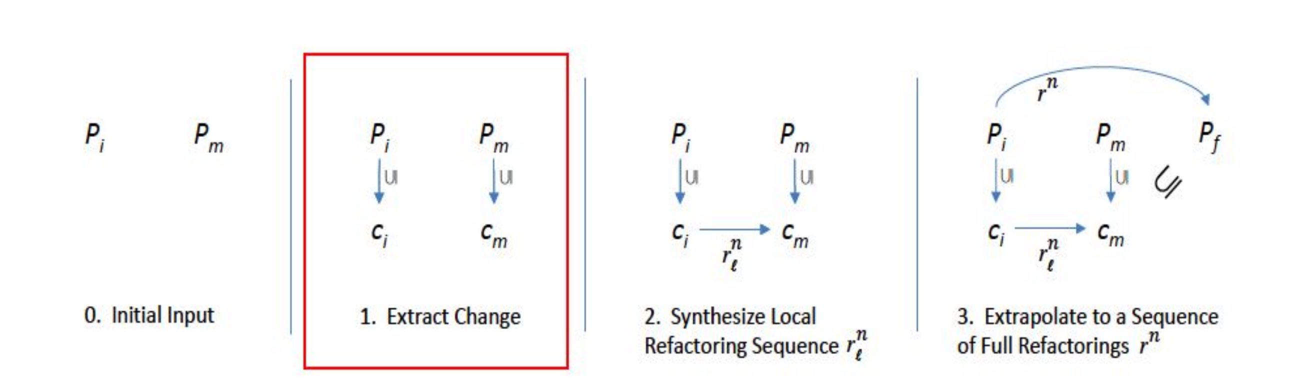


Figure 2. Synthesis Steps

Different pair (Ci, Cm) ?

Abstract Syntax Tree (AST) as a extract function T()

- Ci is a subtree of T(Pi)
- Cm is a subtree of T(Pm)

Example of AST

Pi = x ∗ y + 7

Pm = f() + 7

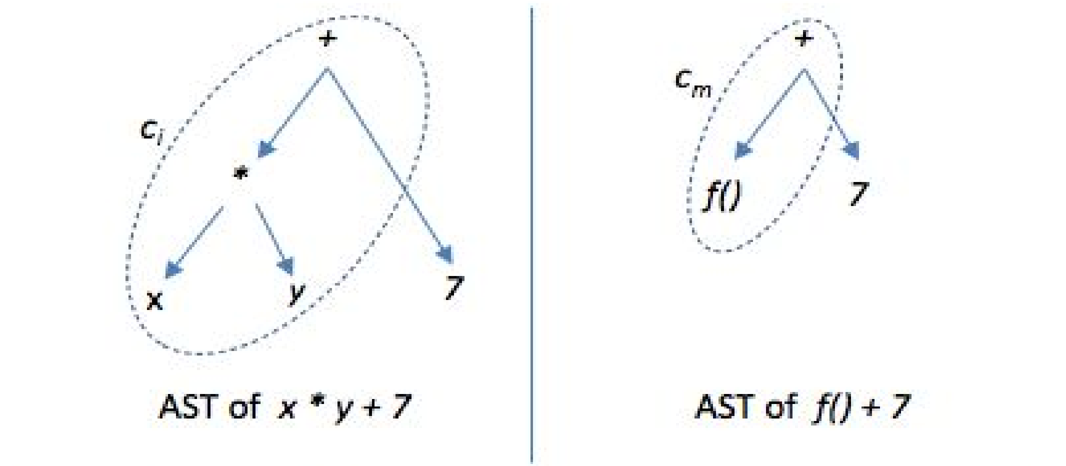


Figure 3. Two ASTs and the change (c_i, c_m) between them. The change is captured with dotted lines.

Synthesize Local Refactoring Sequence

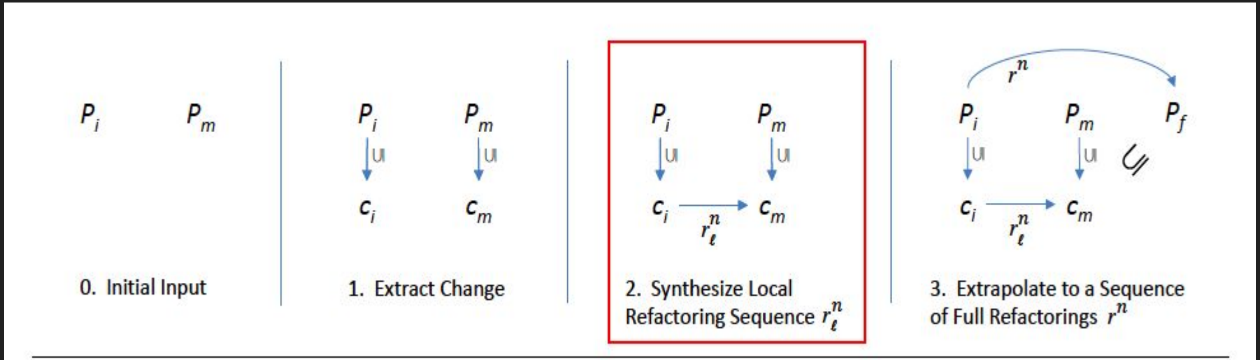


Figure 2. Synthesis Steps

- Discover sequence refactoring
- A* search iteratively computes a distance function d from the initial tree Ci to every other generated tree.
- the search space grows exponentially in the length of desired sequence, that is not effective to use

Extrapolate to Sequence

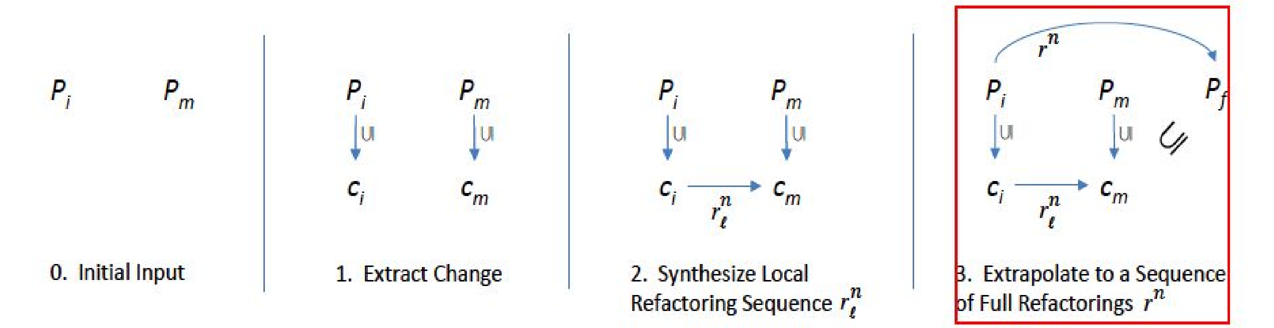


Figure 2. Synthesis Steps

- Obtain a sequence of full refactorings Pf
- If the sequence of full refactorings is infeasible, searches for a different local sequence refractoring and repeats the process.
- Otherwise, obtain the desired program Pf

Specific example

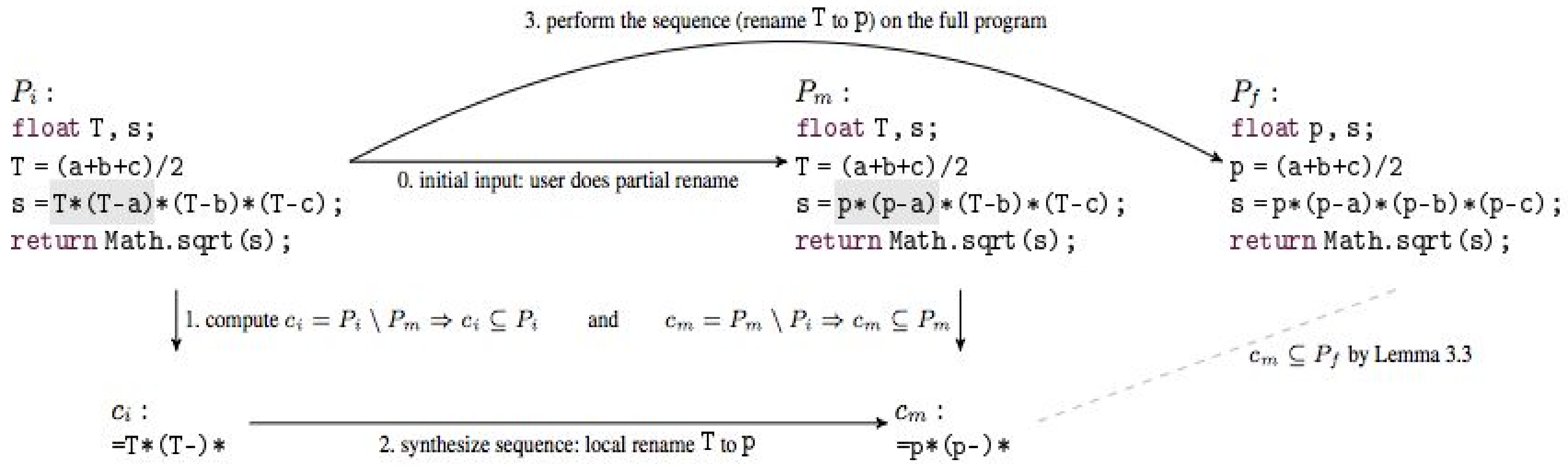


Figure 5. Example of synthesizing a refactoring sequence. Initially (stage 0), the user performs part of the rename (the user change is highlighted in both programs). Then c_i and c_m are computed (stage 1). Then, a sequence of one local rename is discovered (stage 2). Finally, the rename is applied to the full program (stage 3).

Evaluation

Evaluation

These are some of the evaluation that was done:

- Individual Refactoring
- Refactoring Sequences
- Real-world and Synthetic Benchmarks
- User Study

Evaluation

RESYNTH can successfully execute individuals refactorings when given the following edits:

- Rename
- Inline Local
- Inline Method
- Extract Local
- Extract Method with Holes

Evaluation

Refactoring Sequences

- In order for RESYNTH to use the refactoring sequences it has to include a successor function.
- Successors functions takes the current state of the program and produces a finite state of possibles successors states applying refactoring methods
- Since our trees are immutable data structures, it produces a new tree every time, which are added to the search space

Evaluation

Sample of refactoring sequence

Example	steps	Source
ENCAPSULATE DOWNCAST	3	literature [6]
EXTRACT METHOD (advanced)	4	literature [6]
DECOMPOSE CONDITIONAL	6	literature [6]
INTRODUCE FOREIGN METHOD	2	literature [6]
REPLACE TEMP WITH QUERY	3	literature [6]
REPLACE PARAMETER WITH METHOD	3	literature [6]
SWAP FIELDS	3	literature [32]
SWAP FIELD AND PARAMETER	3	literature [25]
INTRODUCE PARAMETER	6	Stack Overflow[9]

Table 1. Realistic examples used to test RESYNTH.

Evaluation

Real-world and Synthetic Benchmarks

- 9 real-world examples
  - 7 correct refactoring
  - 2 equivalent
- 100 random synthetics edits
  - 84% was solved
  - 16 fail (A* would need to explore more than 20000 trees until it could find a refactoring sequence)

Metric	Dataset Real	Dataset Synthetic
Number of tests	9	100
Avg. number of trees searched	87	3752
Avg. number of successors in a search	1296	105310
Avg. search time	0.014s	1.629s
Avg. Eclipse refactoring time	2.953s	1.654s
Refactoring sequence length		
1 refactoring	0	2
2 refactorings	1	45
3 refactorings	5	7
4 refactorings	1	15
5 refactorings	0	3
6 refactorings	2	2
7 refactorings	0	9
8 refactorings	0	0
9 refactorings	0	1
Failure to find sequence after 20000 searched trees	0	16

Table 2. Results for our refactoring sequence search. The A^* heuristic function weights (see Section 3.4) were $a_1 = 0.125$ and $a_2 = 0.25$.

Evaluation

User Study

- Small group of participants (6 elements) with different level of expertise with between 1 and 5 years of java programming experience.
- 3 task
    - RESYNTH
    - Eclipse’s built in refactoring methods
    - Manual editing

Evaluation

Results

- Only 2 were able to find that for task 3 the solution was not quite the expected solution
- 4 found that it is a useful tool
- 1 did not trust the program for a more complex code

Conclusion

RESYNTH is easier to use than the default refactoring tools provided by IDEs because it automates a significant portion of refactoring. However more work needs to done to ensure that RESYNTH works effectively in a complex code base.

Discussion

Discussion Question

RESYNTH is currently being embedded in IDEs like Eclipse.

Do you think a refactoring feature like RESYNTH could be embedded in a language’s compiler so that in can be used outside an IDE and be able to work in a terminal instead?

Discussion Question

Do you think RESYNTH can be used in a complex code base that consists of multiple languages?

Discussion Question

What happen if initial program Pi and modified Pm are not different?

Discussion Question

Why RESYNTH choose to use A* search for synthesizing local refactoring instead of Breadth First Search?

Discussion Question

As a programmer, do you usually use refactoring methods or do you do it by hand? Why?

Discussion Question

If they choose to take out the start refactoring button, what could be another way for RESYNTH to know that code needs to be refactor?